FAKE NEWS CHALLENGE

Stage 1: Stance Classification

Jacob Evans

Data Mining and Statistical Learning

Georgia Institute of Technology

Table of Contents

Background

Recently, claims of “fake news” have ironically made headlines. Emphasized by the 2016 presidential election, the authenticity of once trusted news outlets has been called into question. Without accurate sources of information, the average person left to separate factual news from fiction. The relatively new trend of increased news consumption through digital mediums has further exacerbated this problem by creating a seemingly infinite pipeline of information in which anybody can contribute their “2 cents”. There simply isn’t enough manpower to verify the integrity of every snippet of information posted in the internet – a more scalable solution is necessary if the average-joe is to ever put their faith in what they read on the internet and hear on tv.

Given the recent advancements in natural language processing (NLP), some suggest employing NLP-related machine learning models to alleviate the burden of manually verifying the efficacy news media. True, increases in computation power and the popularization of deep-learning machine learning models have exhibited promising results when working with language data. For example, some deep-learning models have been trained to generate Shakespeare-esque literature all on their own[1] and others to detect cheating on college essays[2], both with high levels of accuracy. It’s no leap of faith then to assume similar results could be realized in the domain of fact-checking news articles.

Fake News Challenge

The objective of the Fake News Challenge (FNC) is to automate the fact-checking process of news articles. With such a lofty goal, the competition has been broken down into more reasonable stages. The first stage relates to stance classification of article headlines and bodies, and will be the focus of this study. Stance classification refers to categorizing the relationship between a headline and an article body into one of four categories: agree, disagree, discuss, unrelated. This task can be further broken down into two separate objectives. The first being a binary classification problem for initially labeling pairs as related or unrelated. Subsequently, a multiclass classification problem labeling all “related” pairs as either agree, disagree or discuss.

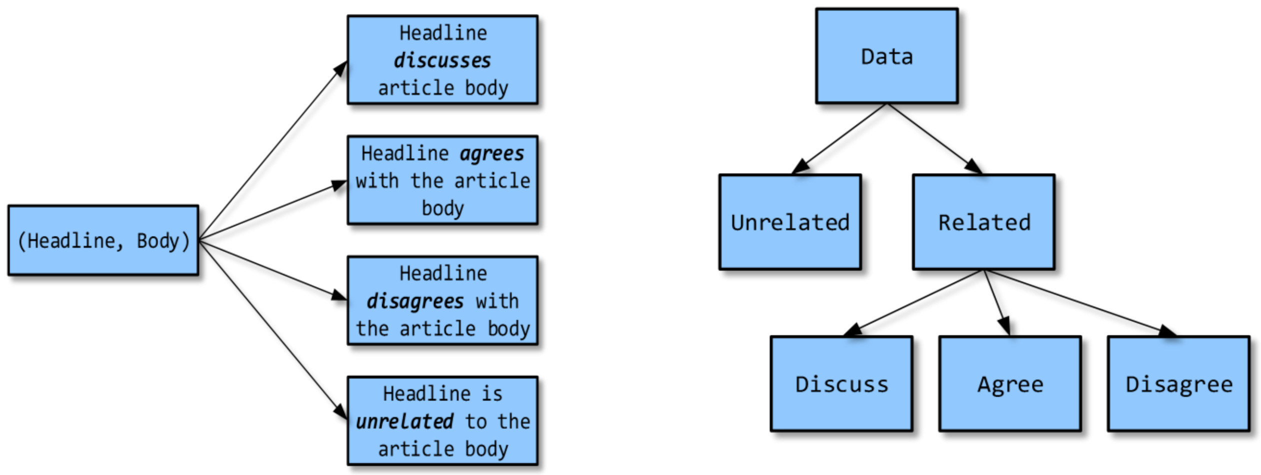

(Headline, Body)
Headline discusses article body
Headline agrees with the article body
Headline disagrees with the article body
Headline is unrelated to the article body
Data
Unrelated
Related
Discuss
Agree
Disagree

Figure 1: Classification diagrams depicting the original stance classification problem (left) and the modified version (right)

[1] https://karpathy.github.io/2015/05/21/rnn-effectiveness/
[2] http://files.eric.ed.gov/fulltext/EJ1064259.pdf

Data

The collection of headlines and article bodies provided by the FNC ultimately takes the following form:

Headline	Body	Stance
"Police find mass graves with…"	"We arrived at the scene at…"	Agrees
"Hundreds of Palestinians flee…"	"Everyone at NATO…"	Disagrees
…	…	…

Table 1: Snapshot of the FNC dataset

There are approximately 1680 original articles in total (i.e. ~1680 (headline, body) pairs). In order to produce the above dataset, the headlines and articles are mixed-and-matched to produce approximately 50,000 records in the final dataset. Following this shuffling procedure, humans manually classified the records' stance. This stance label is considered the "ground truth" for this study. Relying on humans to provide objective labels is problematic and further increases the difficulty of the problem (as the saying goes, "garbage in, garbage out").

The class distribution can be seen in the table below. There is an approximate 25%/75% split between the related and unrelated classes. This class imbalance adds yet another layer of difficulty.

rows	unrelated	discuss	agree	disagree
49972	0.73131	0.17828	0.0736012	0.0168094

Table 2: Class distribution

Scoring

The FNC uses an original scoring procedure to measure the performance of classification models. Accuracy scores are adjusted to reflect the class imbalance present in the dataset. Specifically, correctly classifying unrelated pairs only accounts for 25% of the final accuracy measure while the remaining weight is given to the remaining classes. See the diagram below[3] for clarification.

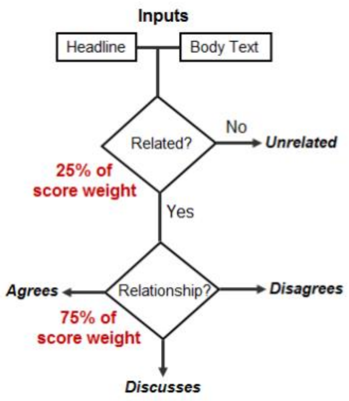


Figure 2: FNC's original scoring diagram

[3] Retrieved from: http://www.fakenewschallenge.org/

Approach

The complexity of the competition necessitates a complex approach. In an effort to leave no stone unturned, we employ models from various domains of academic study including information retrieval, deep-learning, and Bayesian statistics, among others. Our motivation behind this practice is to exploit what each domain does best and then combine it all together to produce a model with truly impressive capabilities.

A high level framework of the prediction procedure can be seen in the figure below. There are three main components: word embedding, binary classification, and multiclass classification. Note several steps in the diagram are omitted due to space constraints including the experimental setup and ensemble methods. See figure A1 in the appendix for an even more detailed diagram.

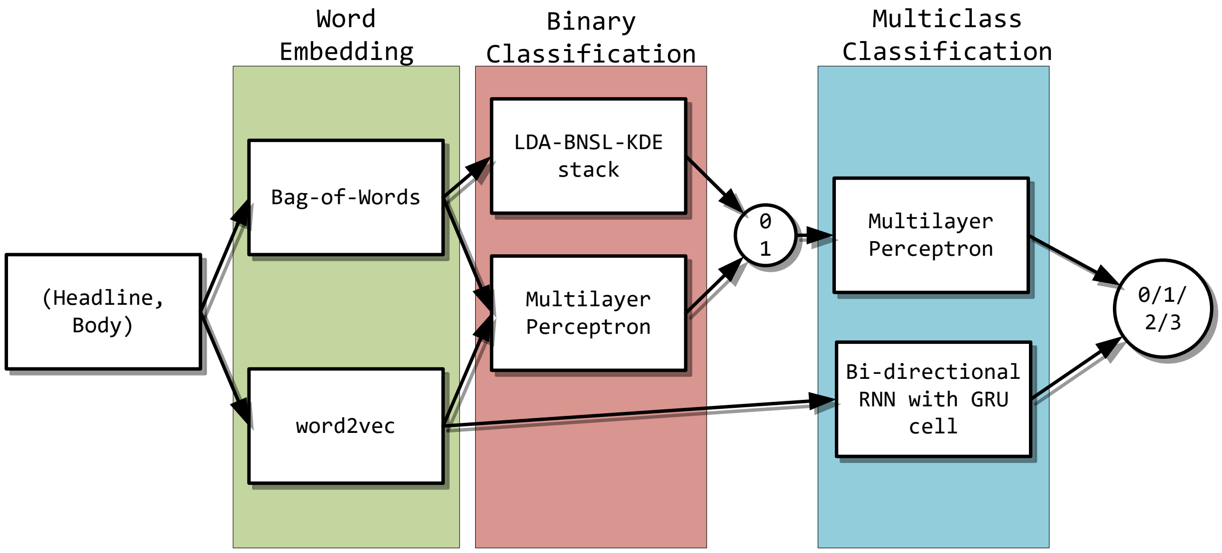


Figure 2: High-level diagram of the modeling approach. Note some components are not explicitly represented such as experiment setup and ensemble methods.

Experimental Setup & Preprocessing

In order to test the validity of our models, we first randomly split the dataset of approximately 50,000 records into the traditional 80/20 proportions to create the training and testing sets, respectively. This is done while ensuring the class distribution across both sets is equivalent to the original distribution (see table 2) to retain consistency. Because there are only around 1680 unique articles, there are some headlines and article bodies that appear in the training set that do not appear in the testing set and visa-versa. Please note the distinction that there are **not** overlapping (headline, body) pairs in the training and testing sets, only overlapping individual headlines and article bodies.

When tuning hyperparameters of the classification models listed below, the training set is split into 5 stratified folds and cross validation is performed. Again, stratified folds preserve the original class distribution in each fold.

Prior to the word embedding procedure, the headlines and article bodies are subjected to gentle preprocessing. This includes converting all letters to lower case and splitting strings of text into lists of words in which each headline or article is a list of list of words. Each nested list represents a sentence in the corresponding headline or article body. As the final step, punctuation is removed. The primary motivation for preprocessing is simply because the word embedding models (described next) require their input to be formatted in a particular manner in practice.

Word Embedding

A challenge unique to this problem is the manner in which the data is provided. Instead of the typical numeric prediction and response variables, the FNC data consists of text. In order to use high-performance machine learning models it's necessary to convert this textual information into a format that these models can easily learn from. This process is referred to as "word embedding". Typically this format is a design matrix in which rows correspond to records (or observations) in the dataset and columns correspond to numeric variables representing a certain attribute of data. To produce such a design matrix, we leverage two well-known word embedding models: Bag-of-Words (BoW) and word2vec (w2v).

Bag-of-Words

The BoW model is one of the simpler word embedding models and yet can produce vector representations of text which encode a substantial amount of semantic information. First, we define the following terms in order to efficiently discuss the model:

- Document – a string of text. In our context this refers to either a headline or an article body
- Vocabulary – the set of unique words present in a corpus of documents[4]
- Corpus – a collection of documents

The training procedure of a BoW model is relatively simple. The model requires a corpus of documents and a mapping scheme as inputs. Following initialization, the model learns the vocabulary of the input corpus and the manner in which it should map subsequent input documents to vector space. That's all there is to it. One can input new documents and the BoW model will output a vector representation of it. The vector output is defined by the training vocabulary and mapping scheme. Specifically, each component of the output vector corresponds to a word in the training vocabulary. The value at each component is calculated based on the original mapping scheme. Several popular mapping schemes include:

- Binary – a 0 or 1 corresponding to if the word in question is present in the input document
- Frequency/Count – the number of occurrences of the word in question in the input document
- Term Frequency-Inverse Document Frequency score – commonly referred to as "tf-idf", this score represents how important the word in question is to the input document given the training corpus

It's important to note that BoW models lose (almost) all contextual information present in a document. That is, information encoded in the sequence of words of the input document is lost. This is why the model is called a "bag" of words – it assumes documents are an unordered set of words rather than a sequence. While this is definitely a shortcoming of this word embedding model, it still manages to produce stellar results depending on the application. For our purposes, we chose the frequency mapping scheme.

Word2Vec

While BoW models are popular and relatively easy to implement, their major design flaw is the loss of contextual information. As a remedy, we turn to Google's Word2Vec which not only offers richer semantic encoding but allows us to capture contextual information as well. Word2vec is a popular deep-learning model which maps individual words to real-valued vectors. Further, it does this *while preserving their semantic meaning*. For example, consider the words "woman" and "royalty". If one adds their word2vec vector

[4] The vocabulary disregards what are known as "stopwords". These are common words which hold little information such as "a", "the", "an", etc. See Porter's "An Algorithm for Suffix Stripping." (1980) for a complete list. Alternatively, see http://www.nltk.org/book/ch02.html

representations together, the result is equivalent to the word2vec vector representation of “queen”. See the figure below for an illustration.

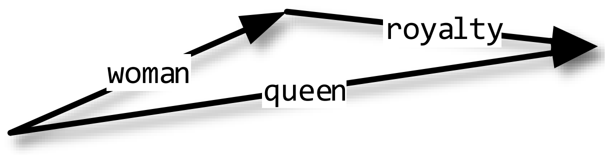


Figure 3: A convincing example of the semantic encoding capabilities of word2vec. The vector addition of “woman” and “royalty” results in the vector “queen”.

This model is an extremely deep and complex neural network, pioneered by some of the greatest minds at Google. Therefore, we won’t delve into the details. At a high-level, the model allows the user to choose the size of the word vectors along with the length of the document (i.e. the number of words of an article to keep).

With such robust semantic encoding capabilities, we assume that sequences of word2vec vectors will not only capture the individual semantic information of word sequences *but also their contextual information*. Luckily, many deep-learning models can handle tensors as input data. Ultimately, this word embedding model outputs a three-dimensional tensor with each dimension characterized as follows:

1. All (headline, body) pairs, i.e. the number of records
2. The length of headline, body sequence of words. First, the article body is appended to the end of the headline, then if the sequence of words is longer than 250, it’s truncated to 250. If it’s less than 250, it’s padded with 0’s.
3. The dimensions of the individual word vectors.

Binary Classification

LDA-BNSL-KDE Stack

Consider a headline about Wall Street and an article body about the war in Syria, because they cover different topics they should intuitively be classified as “unrelated”. The inverse situation also holds true. This simple intuition, that a document’s topics contain the information necessary for the binary classification task, is the motivation for the LDA-BNSL-KDE Stack model. Conceptually, it operates as follows:

1. Learn the topic distribution of the training corpus
2. Featurize the bag-of-words representations of the individual headlines and article bodies using the learned topic distribution
3. Learn the topic hierarchy
4. De-noise the features created in step 2 using the topic hierarchy
5. Fit two kernels to the cosine similarity densities of the related and unrelated (headline, body) pairs in the training set
6. Use these kernels for classification

To complete these tasks, the following (sub)models are used:

- Latent Dirichlet Allocation (LDA)
- Bayesian Network Structure Learning (BNSL)
- Kernel Density Estimation (KDE)

Latent Dirichlet Allocation (LDA)

The LDA model originates from the arena of information retrieval and is routinely used in topic modeling applications. It's a generative model which assumes documents are generated from a *mixture of topics*, where topics are *distributions of words*. The model consists of several parameters, outlined below[5]:

Parameter	**Description**
K	chosen number of topics (an integer)
V	Number of words in the vocabulary
M	Number of documents
N_d	Number of words in document d
α_k	Prior weight of topic k
β_w	Prior weight of word w in a topic
$\phi_{k,w}$	Probability of word w occurring in topic k
$\theta_{d,k}$	Probability of topic k occurring in document d
$z_{d,n}$	Topic of word n in document d
$w_{d,n}$	Word n in topic d

Table 3: LDA parameters

The relationship between these parameters and model's formal definition is,

$$\boldsymbol{\varphi}_{k=1,\ldots,K} \sim Dirichlet_V(\boldsymbol{\beta})\ (distribution\ of\ words\ in\ topic\ k)$$

$$\boldsymbol{\theta}_{d=1,\ldots,M} \sim Dirichlet_K(\boldsymbol{\alpha})\ (distribution\ of\ topics\ in\ document\ d)$$

$$z_{d=1,\ldots,M\ ,n=1,\ldots,N_d} \sim Categorical_K(\boldsymbol{\theta}_d)\ (an\ integer\ between\ 1\ and\ K)$$

$$\boldsymbol{w}_{d=1,\ldots,M,n=1,\ldots,N_d} \sim Categorical_V\left(\boldsymbol{\varphi}_{z_{dw}}\right)\ (an\ integer\ between\ 1\ and\ V)$$

As you can see, this is a hierarchical Bayesian model. Learning the distribution of words for a given number of topics is a problem of Bayesian inference (Blei D. M., 2001). A popular method for solving the inference problem is using collapsed Gibbs sampling (Blei D. M., 2003). A high-level overview is given in figure A5 of the appendix (Chen, 2011).

Upon learning the distribution of words over topics, the LDA model outputs what can be viewed as a "weighted sum of words". For example, when trained on the FNC training set an LDA model initialized to find 30 topics outputs the following for two randomly chosen topics:

Topic A: 0.052*"monday" + 0.032*"domestic" + 0.022*"u" + 0.018*"militants" + 0.017*"two" + 0.017*"would" + 0.015*"terror" + 0.015*"forces" + 0.012*"members" + 0.011*"pentagon"

Topic B: 0.073*"hoax" + 0.061*"news" + 0.040*"dead" + 0.035*"set" + 0.020*"whisperer" + 0.020*"website" + 0.020*"story" + 0.019*"russian" + 0.019*"cesar" + 0.019*"twitter"

In practice, these equations are used to featurize the BoW representations of the training corpus. The result is a design matrix in which the rows represent documents (either a headline or an article body) and the columns represent topics. The values correspond how representative of topic k, document d is. See the diagram below a concise view of how the training corpus is transformed into the design matrix.

[5] Adapted from: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

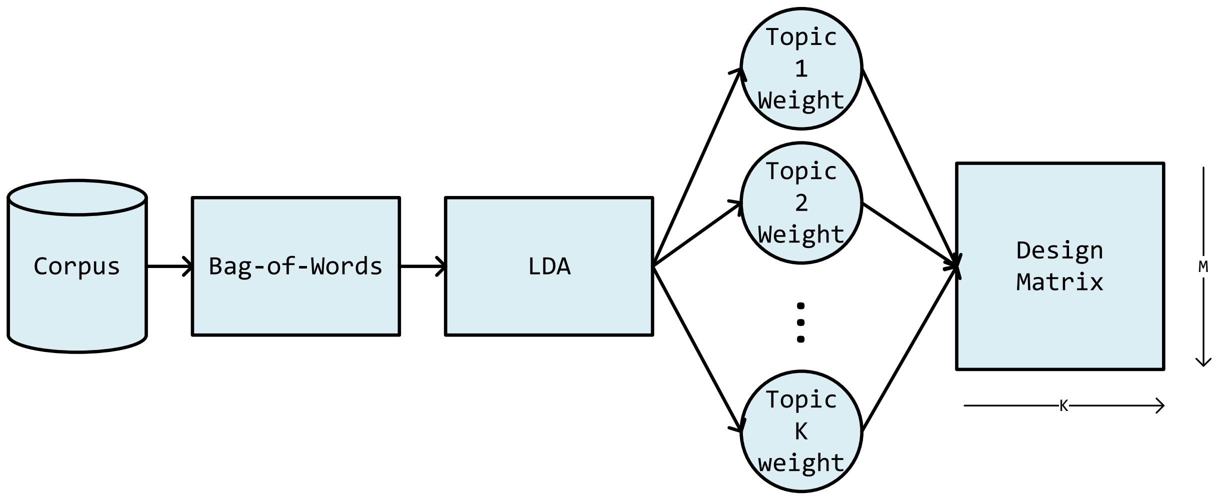


Figure 4: LDA featurization process diagram

For our purposes, we trained an LDA model to find 100 topics in the training set. Choosing the optimal number of topics to find is no easy task given the stochastic nature of LDA and the lengthy training time. If too few topics are chosen, then small differences between topic coverage are lost between documents. On the flip side, if too many topics are chosen then the likelihood of information overlap between topics increases as well. Both outcomes serve to make the binary classification task more difficult by dampening the signal or increasing the noise, respectively.

It should be noted that an important assumption LDA makes is that the topics it finds are independent. Upon closer inspection, this doesn't always hold true. For example, one topic might contain words about politics in general while another contains words specific to the United States' presidential election. Intuitively the second topic should be considered a *subtopic* of the first. We find this theme applies not only to politics, but also to sports, entertainment, etc. Further, it's likely the subtopics contain the discriminatory information necessary for the binary classification task, while the more general "parent level" topics add noise. It's apparent that topics inherently vary in their level of specificity and form a pseudo-hierarchy of information coverage. A natural next step is to formulate a way to exploit this hierarchy.

Bayesian Network Structure Learning (BNSL)

The topic hierarchy intuitively represents a Bayesian network (also called a Bayes net). Bayes nets are directed, acyclic graphs which represent the causal link between random variables. In this context, the random variables are topics of varying specificity (e.g. politics, the 2016 election, etc.). As previously stated, the LDA topics are assumed to be independent – we know nothing about how each topic is related to one another. We turn to Bayesian Network Structure Learning (BNSL) to model the topic relationships. These models attempt to define the causal links between the variables in a design matrix through various means – usually tree "search and score" or approximation methods. While the true Bayes net would be the ideal output of our BNSL model, the learning rate of tree-search algorithms that guarantee this result scale super-exponentially with the number of topics. Instead of waiting for the universe to end, we defer to approximation algorithms (specifically the Chow-Liu trees implementation).

We take a simple approach to exploit this information. Topics which are the "parent" to many "child" topics are simply omitted from the LDA design matrix. We find this to sufficient to dampen the noise generated by the general topics.

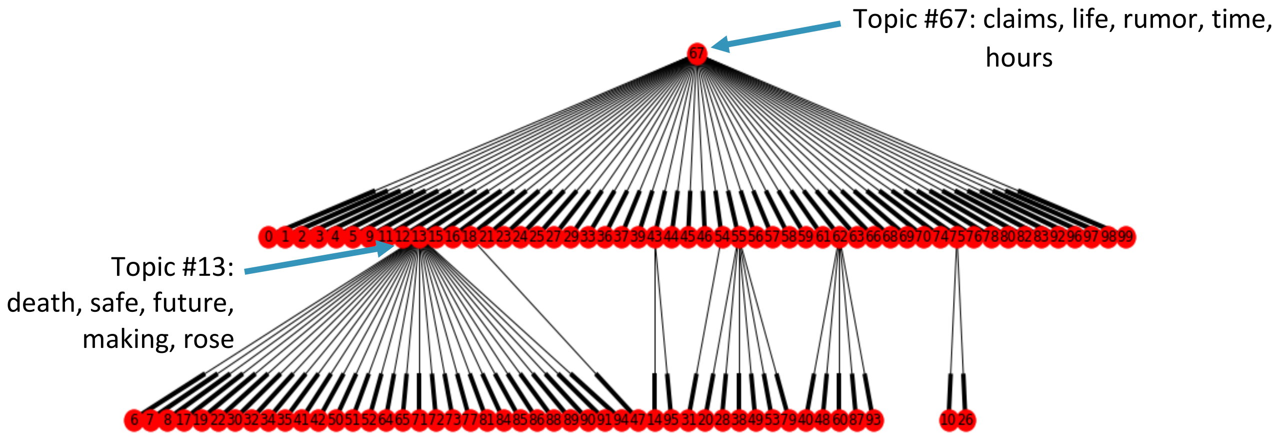

Topic #67: claims, life, rumor, time, hours
Topic #13: death, safe, future, making, rose

Figure 5: BNSL output with the top 5 most representative words for the “dominating” topics

Kernel Density Estimation (KDE)

Having successfully dampened the noise created by the general topics, the next step is to characterize just how similar “related” (headline, body) pairs are to each other (we do the same for dissimilarity between “unrelated” pairs as well). To do so, we measure the cosine of the angle between the topic vectors for each (headline, body) pair for both the “related” and “unrelated” categories and fit a Gaussian kernel to each collection of values. We denote them as the “related” and “unrelated” kernels, respectively.

These kernels are subsequently used to classify (headline, body) pairs in the test set. Specifically, identical steps are taken to measure the cosine similarity between each test (headline, body) pair and then both kernels are evaluated at the measures. The kernel which produces the larger value (interpreted as a higher probability of the pair belonging to that class) is the predicted label for a given test pair. In other words, if the “related” kernel produces a higher value than the “unrelated” kernel, then the test pair in question will be classified as “related”.

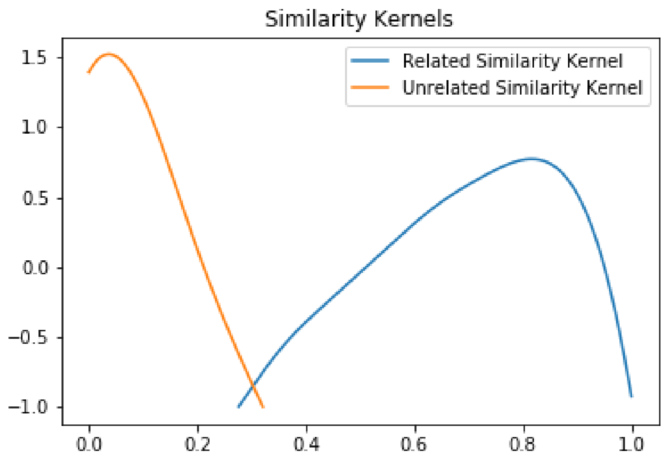


Figure 6: Kernel density estimates used for classification. The x-axis is the similarity score, the y-axis is the density

Note that this procedure is essentially equivalent to a simple linear discriminant analysis that uses the value at which both kernels intersect as the discriminating function.

Multilayer Perceptron (MLP)

Deep-learning models are sometimes considered to be the “holy grail” of natural language processing due to their impressive ability to learn complicated, non-linear relationships. The MLP falls into this category. It’s a neural network with only feed-forward learning capability (as opposed to back-propagation). The primary motivation for choosing this model was the relative success other teams competing in this challenge have seen with it. Further, it’s relatively simple to implement and takes less time to train than many other popular deep-learning models such as LSTMs.

A detailed description of the bag-of-words MLP structure can be seen in figure A7 of the appendix. In total, the model contained 26,948,559 parameters, spread across 7 hidden layers. The activation function for each layer, excluding the final layer, is the “leaky” rectified linear unit function (leaky ReLU). Leaky ReLU is robust against large gradient updates which can sometimes cause ReLU neurons to never activate again during the training procedure (Karpathy, 2014). It’s a safe choice for an activation function. The final layer uses a sigmoid activation function allowing us to interpret the output as a probability of the input belonging to the positive (i.e. “related”) class. A dropout constraint is added to the first layer as a measure against overfitting – only 50% the nodes can be active at once.

The word2vec MLP structure has a similar structure. Again, a detailed description can be seen in figure A8 of the appendix. This MLP contains fewer parameters, only 2,332,882, and has different input dimensions (250, 256) compared to (1, 5000) for the BoW MLP. Recall that 250 refers to the length of the input document and 256 refers to the length of the individual word vectors. In this context, each “document” is structured as the headline vectors, a dividing vector, and the article body vectors. See the figure below for clarification.

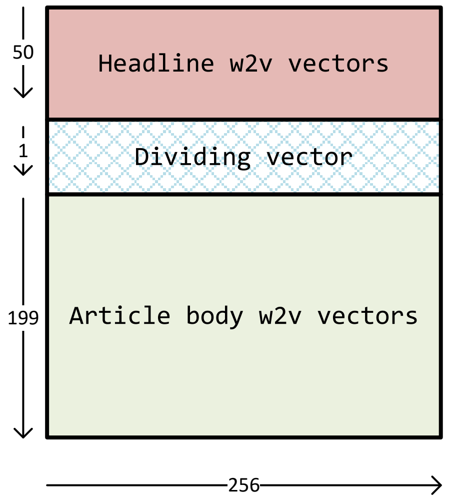


Figure 7: word2vec MLP input dimension visualization

Multi-class Classification

Bi-Directional RNN with GRU Cell

Traditional deep-learning models suffer from the inability to extract information from long sequences of inputs. Even recurrent neural networks, which are designed to capture sequential information, can fall short when the sequence of inputs is sufficiently long. In other words, recurrent neural networks need to be able to learn long-term dependencies from experience. This ability is implemented in bi-directional recurrent neural networks, in which subsequent layers of the network can pass information along to previous layers, updating them with contextual information that might not have been gleamed earlier in the sequence. These networks have special nodes called “gates” which allow for learning long-term dependencies, or in other words, memories.

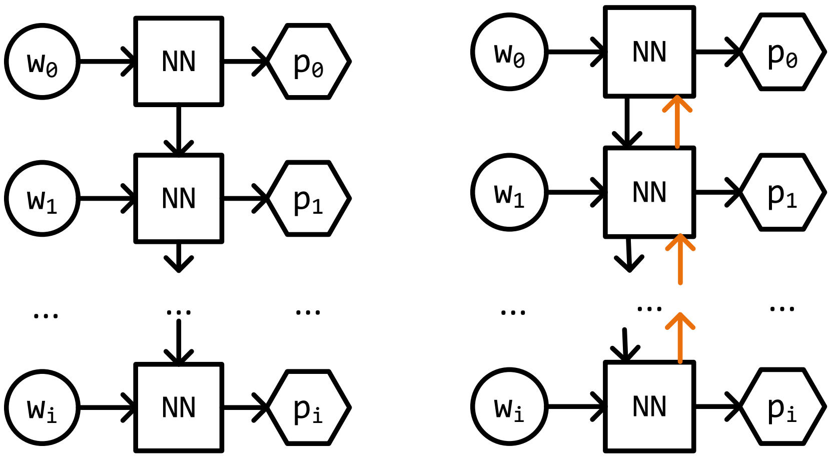


Figure 8: Recurrent neural network (left) and bi-directional recurrent neural network (right)

Bi-directional recurrent neural networks traditionally consist of three layers: the embedding layer, the RNN layer, and the prediction layer. The embedding layer is equivalent to the word2vec model. Essentially sequences of words (i.e. a headline or article body) are mapped to real-valued vectors, or embedded, so that the recurrent neural network can subsequently ingest and learn from them. As previously stated, the output from this layer is a tensor with one dimension defining the number of (headline, body) pairs to train (or test) on, another defining the length of the documents (in words) and the final defining the length of the word vectors.

The RNN layer is where the “magic happens”. In our model, each cell in the hidden layers have what are called “Gated Recurrent Units” (GRU). These enable the RNN to store memories, giving it the ability to learn long term dependencies in sequences of words. For example, consider the incomplete sentence, “I grew up in France, I can speak not only English but also,”. Contextual information implies that the next word should be “French”, and that’s simple for a human to see. GRU cells allow the RNN to learn this contextual information that comes naturally to humans. The training process for the RNN ultimately produces features that the prediction layer will use for classification. We use 200 hidden units in this portion of the model.

The prediction layer is a simple neural network consisting of exactly one hidden layer. It maps the features created from the RNN layer to confidence scores for each of the four potential classification labels. For our purposes the prediction layer is defined to have 100 hidden units. The predicted label is chosen to be the label with the highest confidence score.

Given its modularity and the general complexity of deep-learning models, the bi-directional RNN understandably has numerous hyperparameters which influence its effectiveness. Because of this, the tuning procedure is lengthy and arduous (even more so than MLP). This results in inefficient searching through hyperparameter space when tuning the model, dampening the relative performance of the model overall.

Ensembles

The idea of allowing a committee of weaker learners work together to form a strong learner is appealing for such a complicated problem. We employ a simple ensemble method which allows our set of models to vote on the predicted classification label. Each model’s vote is weighted in such a way that the training error is minimized over 5 stratified folds of cross validation.

Results

LDA-BNSL-KDE Stack

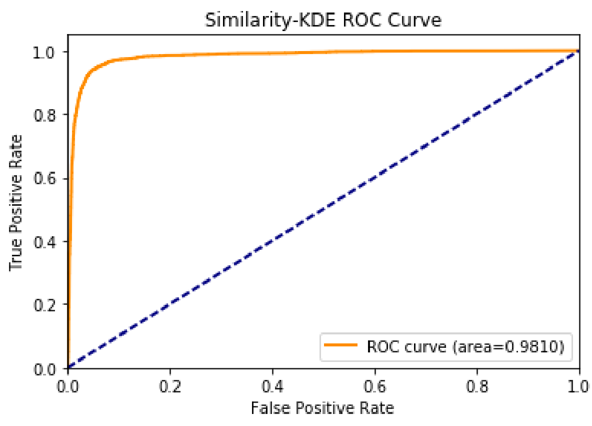
Recall that this model only performs binary classification.

The hyperparameter configuration can be seen in the table below.

Sub-Model	**Hyperparameter**	**Value**
LDA	K	100
BNSL	Fitting-algorithm	'Chow-Liu'
BNSL	Root	67
KDE	Kernel	Gaussian
KDE	Bandwidth	0.06

Table 4: LDA-BNSL-KDE Stack hyperparameter configuration

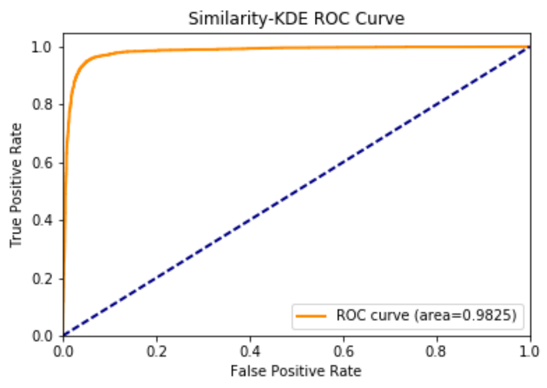
Without BNSL



		Predicted	
		0	1
True	0	6926	438
	1	135	2568

Accuracy: 94.31%

With BNSL



		Predicted	
		0	1
True	0	6988	376
	1	136	2567

Accuracy: 94.91%
(+0.6% lift!)

Figure 9: LDA-BNSL-KDE stack models without BNSL augmentation (left) and with BNSL augmentation (right) ROC curves and confusion matrices

While the BNSL portion of the model might seem like overkill, it produced a measurable lift in accuracy. In fact, it performed exactly as we hoped. There is a decrease in the number of "unrelated" pairs classified as "related".

Multilayer Perceptron

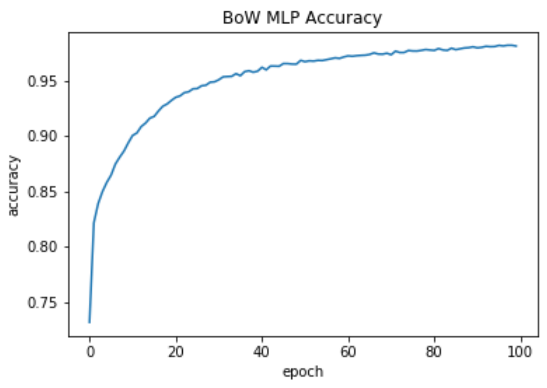
Binary Classification

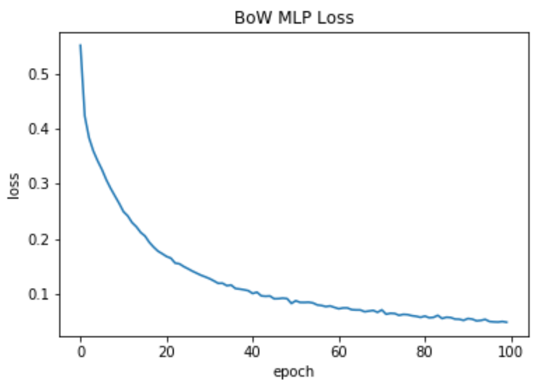
The structure of the network trained on bag-of-words encoded data can be seen in figure A7 of the appendix.

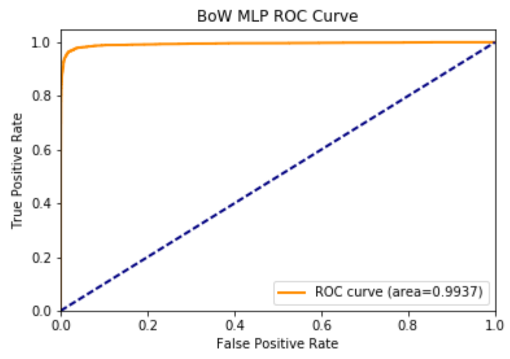
Hyperparameter	Value
# hidden layers	7
# of parameters	26,948,559
Activation function	Leaky ReLU
Dropout	50%
Training Batch Size	32
# Training Epochs	100
Optimization method	ADAM

Table 5: BoW MLP hyperparameter configuration (binary classification task)

To encode the (headline, body) pairs as bags-of-words, we simply append the article body text to the headline text and perform the standard bag-of-words procedure previously described.







		Predicted	
		0	1
True	0	7282	82
	1	199	2504

Accuracy = 97.21%

Figure 10: BoW MLP performance metrics

The loss stabilizes at 100 epochs (loss is a proxy for how well the model performs on training and holdout data). This results in a nice 97.21% accuracy score.

The structure of the network trained on word2vec encoded data can be seen in figure A8 of the appendix.

Several important hyperparameters are listed below.

Hyperparameter	Value
# layers	9
# of parameters	64,842,919

Activation function	Leaky ReLU
Dropout	50%
Training Batch Size	32
# Training Epochs	100
Optimization method	ADAM

Table 6: w2v MLP hyperparameter configuration (binary classification task)

The following plots illustrate the rate of learning over training epochs.

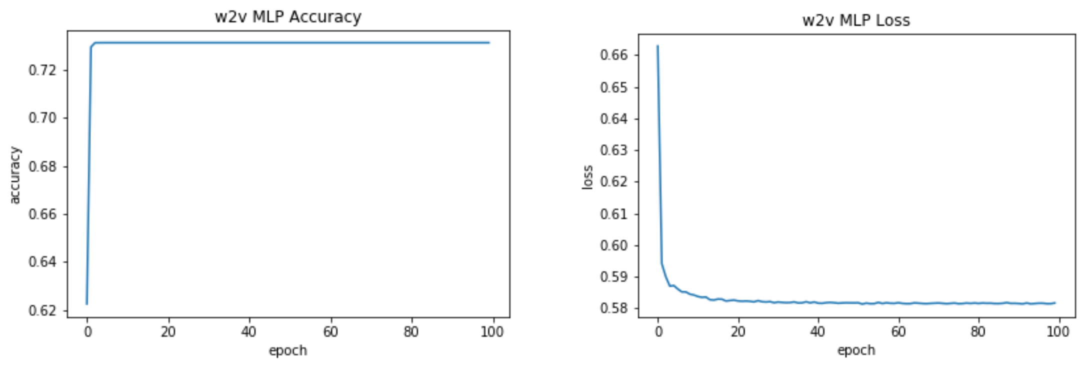


Figure 12: word2vec MLP training metrics

After ~20 epochs the loss stabilizes. Unfortunately, the overall accuracy is just ~73%, this is exactly what we would expect if flipping a coin with outcomes equal to the proportion of the "related" and "unrelated" classes.

Multi-class Classification

The structure for the BoW MLP trained for multi-class classification is almost identical to the structure of the network trained for the binary classification task. The only difference is the output dimensions, instead of 1, there are 3 output nodes, one for each class ("agree", "disagree", and "discuss"). Because of their similarity, the hyperparameter configuration is omitted.

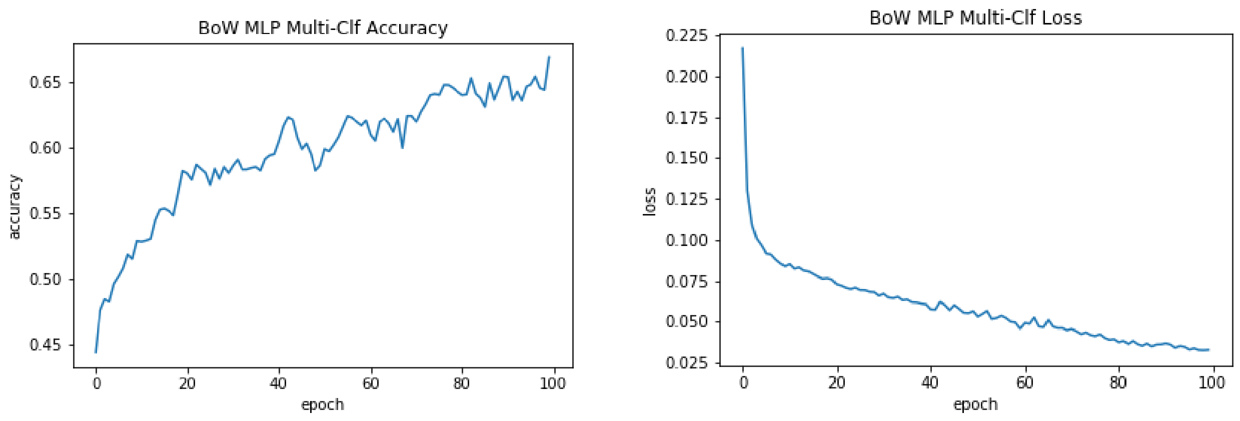


Figure 13: BoW MLP training metrics (multi-class classification task)

		Predicted		
		agree	disagree	discuss
True	agree	678	42	0
	disagree	48	123	0
	discuss	1020	434	358

Figure 14: BoW MLP confusion matrix (multi-class classification task)

Accuracy = 42.88%

Unfortunately, the BoW model doesn't have stellar results when trained on the multi-class classification data.

Similarly, the structure for the w2v MLP trained for multi-class classification follows an almost identical structure to the network trained for the binary classification task. Again, the difference is in the output dimensions, following the same scheme as the BoW MLP.

		Predicted		
		agree	disagree	discuss
True	agree	720	0	0
	disagree	171	0	0
	discuss	1812	0	0

Figure 15: w2v MLP confusion matrix (multi-class classification task)

Accuracy = 26.63%

Once again, the w2v MLP fails to impress. The poor performance is likely attributable to the extremely small corpus of documents the w2v model has to learn from (only "agree", "disagree", and "discuss" pairs, less than 25% of the original dataset), decreasing its ability to encode semantic information.

Bi-Directional RNN with GRU Cell

The hyper parameter configuration can be seen in the table below.

Hyperparameter	**Value**
Headline length to keep (words)	50
Article Body length to keep (words)	200
Type of Unit	GRU
# Hidden Units in RNN	200
# Layers in Prediction Layer	1
# Training Epochs	10
Training batch size	32
Optimization method	Stochastic Gradient Descent

Table 7: Bi-directional RNN with GRU Cell hyperparameter configuration

		Predicted			
		agree	disagree	discuss	unrelated
True	agree	647	0	30	43
	disagree	144	0	15	12
	discuss	99	0	1613	100
	unrelated	113	0	86	7165

Figure 16: Bi-directional RNN with GRU cell confusion matrix

Runtime: 29866.80 seconds ≈ 7 hours

Accuracy: 90.74%

Regarding the multi-class classification task, the bi-directional RNN beats out the other MLP models by a wide margin. This is more or less expected given the capabilities of the model (not to mention the amount of time it took to train!).

Ensembles

The optimal vote weights for the binary classification task can be seen in the table below:

Model	**Vote Weight**
LDA-BNSL-KDE Stack	0.0
BoW Multilayer Perceptron	1.0
w2v Multilayer Perceptron	0.0
Bi-directional RNN with GRU Cell	0.0

Table 8: Optimal voting weights for the binary classification task's ensemble

Given the overwhelming accuracy of the BoW MLP model, it's no surprise that the optimal weighting scheme defers all binary classification decisions to it. The result is an overall binary classification accuracy score equivalent to the BoW MLP's accuracy: 97.21%.

The classification vote weights for the multi-class classification task can be seen in the table below:

Model	**Vote Weight**
BoW Multilayer Perceptron	0.0
w2v Multilayer Perceptron	0.0
Bi-directional RNN with GRU Cell	1.0

Table 9: Optimal voting weights for the binary classification task's ensemble

We see similar results in the multi-class classification context. Given the success of the bi-directional RNN with GRU cell, it's again no surprise the optimal weighting scheme favors this model. The results aren't particularly exciting to view, but the accuracy measures don't lie[6]. The final confusion matrix is as follows:

		Predicted			
		agree	disagree	discuss	unrelated
True	agree	608	0	28	84
	disagree	139	0	12	20
	discuss	96	0	1563	153
	unrelated	4	0	12	7348

Figure 17: Overall confusion matrix from final the final ensemble

Overall accuracy: 94.56%

Using the FNC's official scoring scheme as previously described, the overall FNC score is:
0.25*7348 + 0.75*(608 + 0 + 1563) = 1780.25 points out of 3868.25 possible points.

[6] *While this soft voting ensemble section could have been omitted in favor of "the BoW MLP handles binary classification, the bi-directional RNN handles the rest", the optimization framework was already in place before these weights were determined. Therefore, we opted to keep this section.*

Findings

Regarding the binary classification task, the LDA-BNSL-KDE stack produced the highest accuracy suggesting that further research into topic modeling is warranted. This result is intuitive due to the nature of the problem.

While deep-learning models tend to be the “go-to” models for NLP applications, the necessity for computing power cannot be understated. It’s interesting to note how the BoW MLP model out-performed the w2v MLP model. It’s likely this is a direct result of how the data was encoded. While word2vec is lauded as one of the best word encoding models, the corpus of documents it is traditionally trained on are massive. The example in figure 3 is the result of the word2vec model trained on the Google News corpus containing some 3.5 billion news articles. Compared to our paltry 1680 articles, it’s unsurprising the amount of semantic information captured suffers.

Secondary Applications

The binary classification models could be used for clickbait detection given their high accuracy detecting when two strings of text are unrelated to each other. Because click-through rate is one of online advertising’s driving metrics, clickbait has become a fairly common occurrence. Sure, not all clickbait advertisements contain completely different information between the link and the resulting content, but in the instances that they are completely different, our model should be able to detect it with a relatively high accuracy (given a large enough training set).

Future Work

Extensive hyperparameter tuning is, in our opinion, the first area that needs more exploration. Due to the long training times for all the models and the amount of hyperparameters to tune, the hyperparameter search space hasn’t been as well explored as it should be. We estimate that given enough time and computing resources, the accuracy of the models could be greatly improved without adding any more complexity to the current framework.

Apart from tuning the hyperparameters further, we believe that implementing sentiment analysis would improve the multi-class classification task. Intuitively, when an article is discussing a topic, the language used is less “emotionally charged” than those that either agree (positively charged) or disagree (negatively charged). The complexity here arises from the lack of a ground-truth set of “emotionally charged” words that could be leveraged for such a task. While such sets do exist, most are manually curated by humans which is something that should be avoided if possible to mitigate the introduction of bias.

In regards to the LDA-BNSL-KDE stack, we recognize that topic independence should be dealt with before the modeling process, as opposed to “hacking” the LDA model’s output to regain some of the information potentially lost through the independence assumption. Fortunately, hierarchical topic models exist, even a hierarchical LDA model. However, consensus seems to be that hierarchical LDA produces less coherent topics compared to vanilla LDA. Therefore, we posit that our current method is healthy middle ground between the two, but that other methods are still worth exploring.

Finally, since topic modeling performed so well on the binary classification task, using the topic representativeness design matrix created in the LDA-BNSL-KDE stage (see figure 4) as supplemental features for the deep-learning models should produce better results.

Academic Takeaways

First and foremost, the most important lesson learned is the difficulty of working with unstructured data. The process of encoding information into a design matrix that’s ingestible by machine learning models seems to be just as important (if not more important) than which classification models to use. Further, the computational power required to employ the more cutting-edge models like bi-directional RNNs with GRU cells cannot be understated. All the experiments were run on a powerful computer (16GB RAM, i5 processor overclocked @ 4.5 GHz, with GPU-enabled tensorflow) and yet most of the time was spent simply waiting for results. This made the task of tuning hyperparameters extremely slow and frustrating.

At the end of the day though, this project was fun to work on. The feeling of going from a jumble of unstructured data to an ROC curve that isn’t horrible is pretty great. Because this is a competition hosted outside of the academic arena, more work will be done to fine-tune the models even further. The goal is to be one of the top three teams (they receive cash prizes).

Appendix

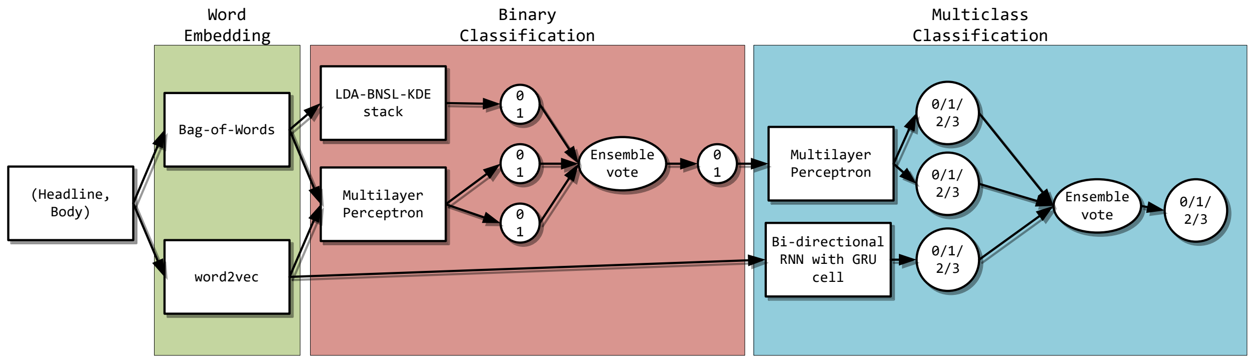


Figure A1: Complete approach framework diagram

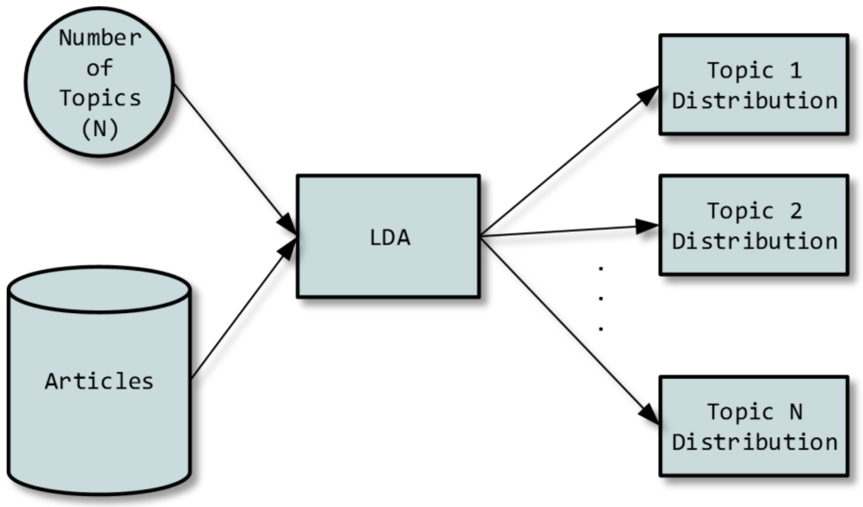


Figure A2: Simplified LDA process flow

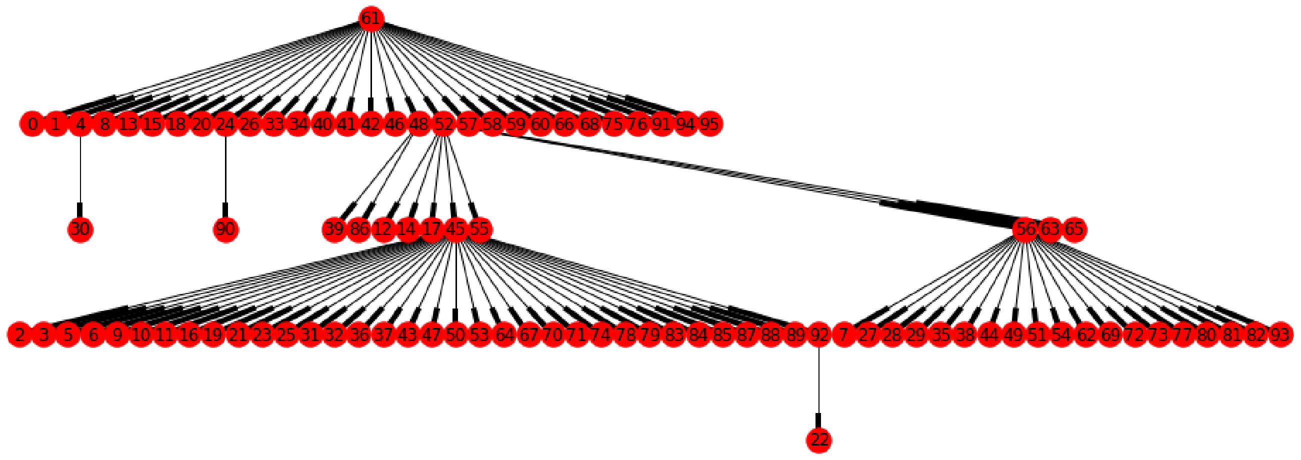


Figure A3: Topic BNSL after several rounds of pruning the parent-level topic nodes

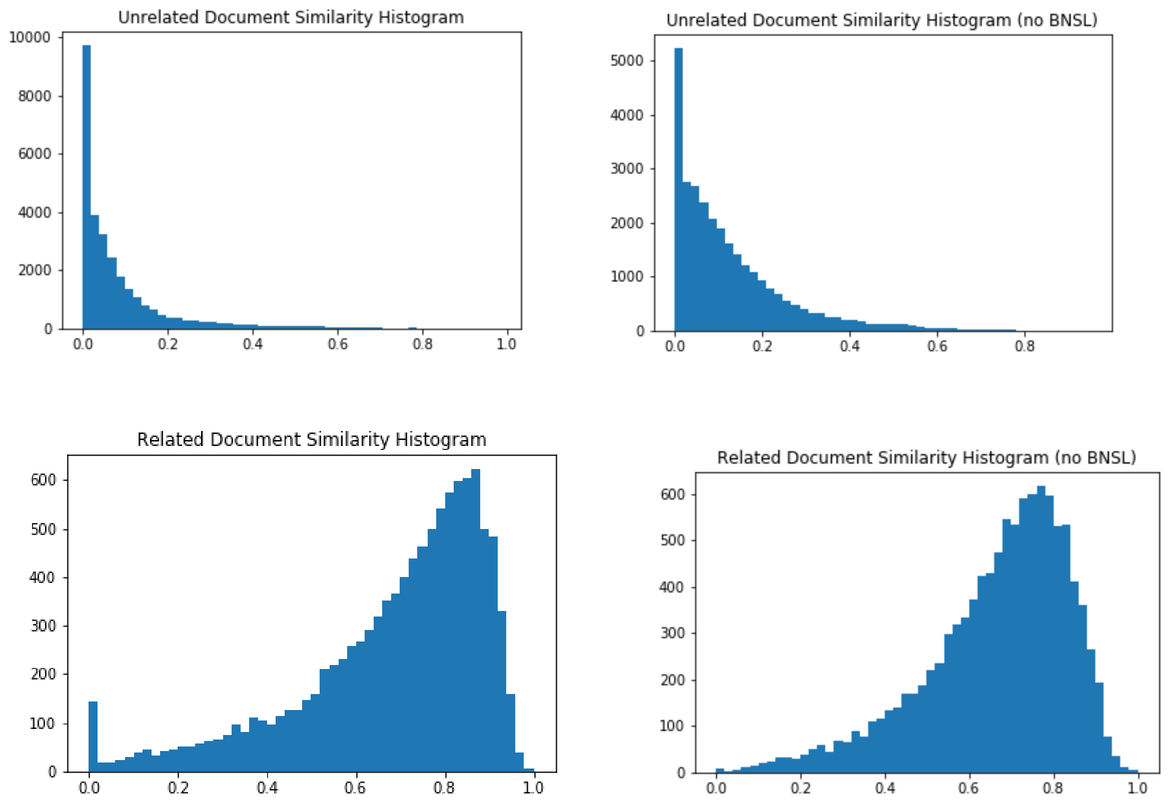


Figure A4: Similarity score histograms for the "related" and "unrelated" classes shown with the BNSL step (removing the parent-level topics) (left) and without the BNSL step (right)

Learning Topic Distributions via Collapsed Gibbs Sampling

```
1. Initialization step:
   For each document d = 1,…,M
       For each word n = 1,…,$N_d$ in document d
           Assign $w_{d,n}$ to topic k with probability 1/K
2. Learning step:
   For each document d = 1, …, M
       For each word n = 1, …, $N_d$ in document d
           For each topic k = 1, …, K
               Compute $P(k \mid d) = \frac{\sum_{w=1}^{N_d} 1(\boldsymbol{z}_{d,n}=k)}{N_d}$, the proportion of words in
               document d that belong to topic k

               Compute $P(w_{d,n} \mid k) = \frac{\sum_{d=1}^{N} 1(\boldsymbol{z}_{d,w}=k)}{\sum_{d=1}^{N} |w_{d,n} \in d|}$, the proportion of assignments
               to topic k over all documents for the word $w_{d,n}$

               Set $w_{d,n} = argmax_k\{P(k|d) * P(w_{d,n}|k)\}$, where $P(k|d) * P(w_{d,n}|k)$ is the
               probability that topic k generated $\mathbf{w_{d,n}}$
3. Repeat step 2 sufficiently many times until steady state is achieved
```

Figure A5: LDA learning algorithm

Given the model definition, the conditional probabilities are:

$$P(w_{d,n} = v \mid W_{-(d,n)}, Z, \boldsymbol{\beta}) \propto |W_{v,k,-(d,n)}| + \beta_v$$

$$P(z_{d,n} = k \mid Z_{-(d,n)}, w_{d,n} = v, W_{d,n}, \boldsymbol{\alpha}) \propto (|Z_{k,d,-(d,n)}| + \alpha_k) * P(w_{d,n} = v \mid W_{-(d,n)}, Z, \boldsymbol{\beta})$$

The first conditional probability describes the probability the word designated by token v is in the nth position of document d, given the all other words, their topic assignments, and the beta vector (a hyperprior). The second conditional probability describes the probability of that word belonging to topic k, given the topic assignments of all other words, the word itself, the set of all words, and the alpha vector (a hyperprior).

Figure A6: LDA posterior probabilities

```
_________________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 5000)              0
_________________________________________________________________
dense_10 (Dense)             (None, 5000)              25005000
_________________________________________________________________
batch_normalization_9 (Batch (None, 5000)              20000
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 5000)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5000)              0
_________________________________________________________________
```

```
dense_11 (Dense)             (None, 312)               1560312   
_________________________________________________________________
batch_normalization_10 (Batc (None, 312)               1248      
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 312)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 312)               97656     
_________________________________________________________________
batch_normalization_11 (Batc (None, 312)               1248      
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 312)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 312)               97656     
_________________________________________________________________
batch_normalization_12 (Batc (None, 312)               1248      
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 312)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 312)               97656     
_________________________________________________________________
batch_normalization_13 (Batc (None, 312)               1248      
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 312)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 156)               48828     
_________________________________________________________________
batch_normalization_14 (Batc (None, 156)               624       
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)   (None, 156)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 78)                12246     
_________________________________________________________________
batch_normalization_15 (Batc (None, 78)                312       
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)   (None, 78)                0         
_________________________________________________________________
dense_17 (Dense)             (None, 39)                3081      
_________________________________________________________________
batch_normalization_16 (Batc (None, 39)                156       
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 40        
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0         
=================================================================
Total params: 26,948,559.0
Trainable params: 26,935,517.0
Non-trainable params: 13,042.0
_________________________________________________________________
```

Figure A7: Detailed BoW MLP structure (binary classification task)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (InputLayer)         (None, 250, 256)          0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 64000)             0         
_________________________________________________________________
dense_28 (Dense)             (None, 1000)              64001000  
_________________________________________________________________
batch_normalization_24 (Batc (None, 1000)              4000      
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 1000)              0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_29 (Dense)             (None, 500)               500500    
```

```
_________________________________________________________________
batch_normalization_25 (Batc (None, 500)               2000
_________________________________________________________________
leaky_re_lu_25 (LeakyReLU)   (None, 500)               0
_________________________________________________________________
dropout_8 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_30 (Dense)             (None, 250)               125250
_________________________________________________________________
batch_normalization_26 (Batc (None, 250)               1000
_________________________________________________________________
leaky_re_lu_26 (LeakyReLU)   (None, 250)               0
_________________________________________________________________
dense_31 (Dense)             (None, 250)               62750
_________________________________________________________________
batch_normalization_27 (Batc (None, 250)               1000
_________________________________________________________________
leaky_re_lu_27 (LeakyReLU)   (None, 250)               0
_________________________________________________________________
dense_32 (Dense)             (None, 250)               62750
_________________________________________________________________
batch_normalization_28 (Batc (None, 250)               1000
_________________________________________________________________
leaky_re_lu_28 (LeakyReLU)   (None, 250)               0
_________________________________________________________________
dense_33 (Dense)             (None, 250)               62750
_________________________________________________________________
batch_normalization_29 (Batc (None, 250)               1000
_________________________________________________________________
leaky_re_lu_29 (LeakyReLU)   (None, 250)               0
_________________________________________________________________
dense_34 (Dense)             (None, 62)                15562
_________________________________________________________________
batch_normalization_30 (Batc (None, 62)                248
_________________________________________________________________
leaky_re_lu_30 (LeakyReLU)   (None, 62)                0
_________________________________________________________________
dense_35 (Dense)             (None, 31)                1953
_________________________________________________________________
batch_normalization_31 (Batc (None, 31)                124
_________________________________________________________________
leaky_re_lu_31 (LeakyReLU)   (None, 31)                0
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 32
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 64,842,919.0
Trainable params: 64,837,733.0
Non-trainable params: 5,186.0
_________________________________________________________________
```

Figure A8: Detailed word2vec MLP structure (binary classification task)

References

Blei, D. M. (2001). Latent Dirichlet Allocation. *Neural Information Processing Systems.* Vancouver, British Columbia, Canada: MIT Press. Retrieved from Neural Information Processing Systems.

Blei, D. M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research 3*, 993-1022.

Britz, D. (2015, 10 27). *RECURRENT NEURAL NETWORK TUTORIAL, PART 4 – IMPLEMENTING A GRU/LSTM RNN WITH PYTHON AND THEANO*. Retrieved from WildML: http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

Chen, E. (2011, 08 22). *Introduction to Latent Dirichlet Allocation*. Retrieved from Edwin Chen: http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/

Dirichlet-Multinomial Distribution. (2017, 02 24). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution

Karpathy, A. (2014, 10 22). *Convolutional Neural Networks for Visual Recognition*. Retrieved from Github: http://cs231n.github.io/neural-networks-1/

Latent Dirochlet Allocation. (2017, 04 08). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Ma, J. (2016, 04 02). *All of Recurrent Neural Networks*. Retrieved from Medium: https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e

Olah, C. (2015, 08 07). *Colah's Blog*. Retrieved from Github: http://colah.github.io/posts/2015-08-Understanding-LSTMs/